D0 Data Requests / Settings

System Implementation

Mar 4, 1990

Introduction

The new message formats for D0 data requests / settings, described in the document "D0 CDAQ Network Data Transmission Protocol" by Alan Jonckheere, use the Acnet header designed by Charlie Briegel to support generalized task-task communications across a network. The Network Layer software in the VME Local Stations supports these Acnet header-based messages. This note describes the implementation of the support for the new data request and setting messages.

Message flow

When a request or setting message is received, it is directed to a well-known taskname RPYR. At initialization, the DZero Request Task creates a message queue (called DREQ) that is used to receive Acnet header-based messages directed to the taskname RPYR. NetCnct registers this taskname to the Network Layer.

```
   Function NetCnct (taskName, queueId, eventMask, VAR taskId);
```

The eventMask is left zero, as the Request Task will simply wait on the message queue rather than wait on an event. The Request Task then enters an infinite loop that calls NetCheck to wait for a message and, upon receiving one, process it.

```
   Function NetCheck (taskId, timeOut, VAR msgRef);
```

When the function returns with valid status, the message type is checked as found in the first word of the Acnet header. If it is a USM (unsolicited message) with the CAN bit set, the request identified by the message id is cancelled. If it is a request message type, the message following the header (and the format block) is checked. If it is a setting, it is processed immediately. If it specifies a request for data, then a set of 3 message blocks are allocated for support of the new request. (If the request specifies an existing active message id, then the existing request is cancelled.) The basic request block houses the various parameters needed to monitor the request activity. Two pointers are included in that block that point to the other related allocated blocks—the internal ptrs block and the answers block.

The basic DZero request block (type #13) contains the array of listype control blocks (LCBs) and the period specification.

$00	MBlkSize	LinkNext		MBlkType=13
$08	ReqAHdr	ReqAHdr#by	Request	Request#by
$10	Ptr to internal ptrs block (#14)		Ptr to answers block (#9)	
$18	pSpec	pSpec#by	Update Cntr	
ReqAHdr=$20	flags	status	destNodeLan	srcNodeLan
$28	destTaskName		srcTaskId	msgId
$30	msgLng	pSpecCur	pSpecDly	pSpecBlk
Request=$38	8 2 0 8	pSpec	sData	#listypes
$40	listype#	dOffset	#bytes	#idents
$48	identLng	idents	other	…
pSpec	period specification			…

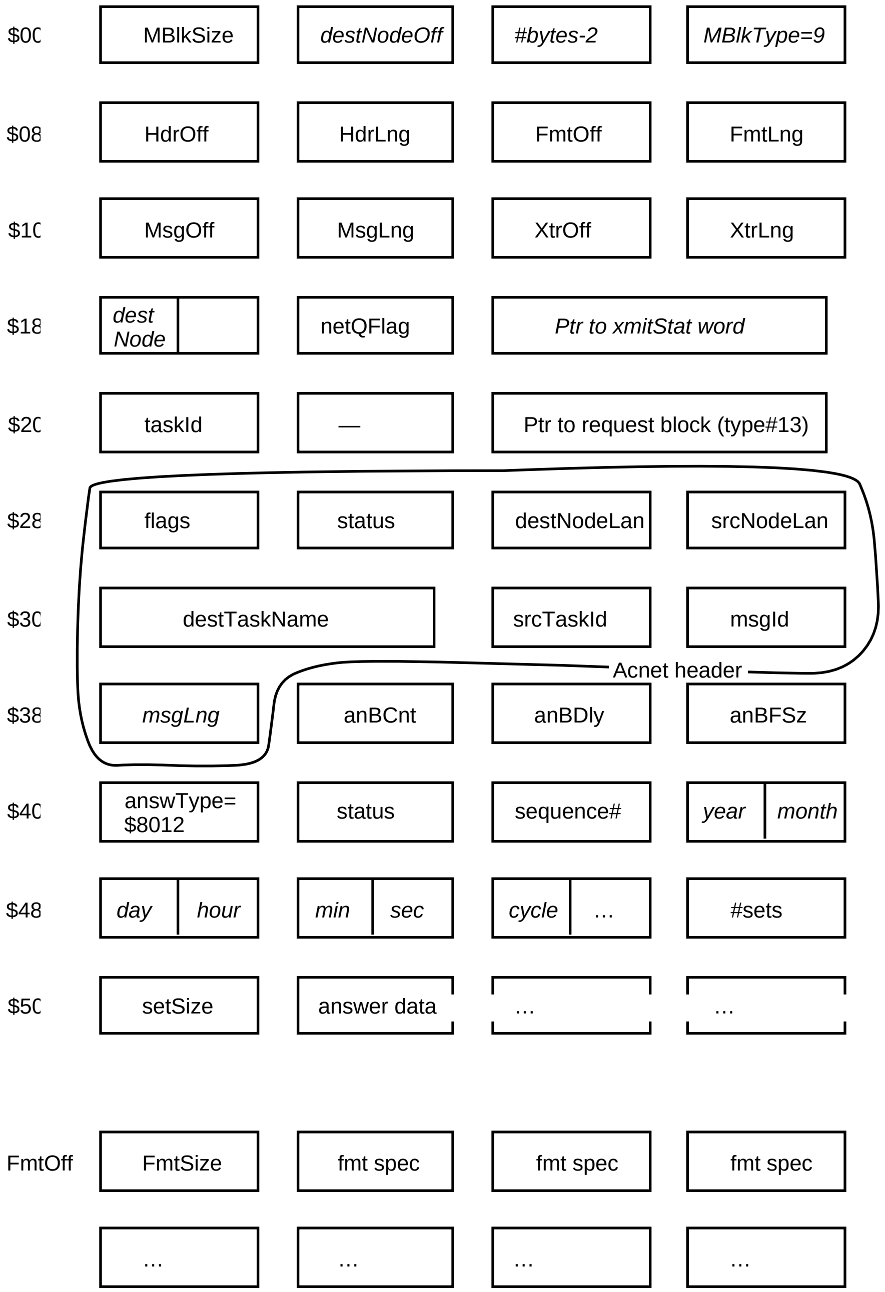
Acnet header

listype
control
block
(7 words)

The Internal Ptrs block (type #14) contains the array of internal ptrs that are used to update the request (build the answers) efficiently.

$00	pBlkSize	ptrsOff=8	nPtrs	pBlkType
ptrsOff	intPtr1		intPtr2	
	…			

The answers block (type #9) is an Acnet message block of the form used by the Network Layer software when the answers are to be returned to the requesting node/task. It also includes a pointer to the parent request block (type #13) for use by QMonitor for one-shot requests that need automatic cancellation.



$00	MBlkSize	*destNodeOff*	*#bytes-2*	*MBlkType=9*
$08	HdrOff	HdrLng	FmtOff	FmtLng
$10	MsgOff	MsgLng	XtrOff	XtrLng
$18	*dest Node* |	netQFlag	*Ptr to xmitStat word*	
$20	taskId	—	Ptr to request block (type#13)	
$28	flags	status	destNodeLan	srcNodeLan
$30	destTaskName		srcTaskId	msgId
$38	*msgLng*	anBCnt	anBDly	anBFSz
$40	answType= $8012	status	sequence#	*year* | *month*
$48	*day* | *hour*	*min* | *sec*	*cycle* | …	#sets
$50	setSize	answer data	…	…
FmtOff	FmtSize	fmt spec	fmt spec	fmt spec
	…	…	…	…

Request message processing also includes building the format block for inclusion in the answer response message. To do this there is a format specification template for each listype included in the LTT module. The template is scanned according to the #bytes of data requested per ident. If the end of the template is reached, and the #bytes requested is not exhausted, the request is in error. This constitutes a new restriction on data requests, where the #bytes that can be requested using a given listype is constrained according to the format spec

After the request support blocks have been filled, the basic request block is inserted into the chain of active data requests using INSCHAIN. It is inserted at a position adjacent to another request block made by the same node, if any, in order to increase the likelihood of combining the answer responses of multiple requests into the same network frames. Then the Update Task is triggered to update the request and build the first set of answers immediately.

The request message is processed as it resides in the network frame input buffer DMA'd into memory by the chipset. This processing includes "compiling" the request into the internal ptrs array for later update processing. The message count word in the network frame buffer is decremented to signal to the network that the request message space is now free for future use. Note that initializing the request as it resides in the network buffer (instead of using NetRecv to copy it into the caller's buffer) saves copying the ident arrays in the request, at the expense of the additional responsibility of decrementing the message count word when finished with the request message. Of course, both the LCBs and period specification must be copied into the request block for later update processing.

Updating requests

The Update Task scans through all active requests each cycle to update any which are due for processing. It checks for this new request block type (#13) and builds the answers accordingly. The read-type routines are called for each listype using the array of internal pointers to build the answer data. Other data must be placed into the header of the answer message. The format block, however, should remain constant for the request's activity.

A facility for blocking answer responses is specified in the new protocol's period specification. The two parameters given are the maximum number of messages to build before responding and the timeout delay before responding when the maximum number of answers have not yet been built. The size of the answers block is affected by the maximum number of messages parameter, as it lengthens both the format block as well as the answer message itself. As a result, an estimate of the size of the returned answers and the required format block with any blocking is needed before the answers block (type #9) can be allocated.

When the Update Task has built answers that are to be returned to the requester, it invokes the NetQueue routine to do it. Just before that, however, it calls NetXChk to flush any existing queued messages that are going to a different node or use a different protocol type (different SAP) to the network chipset. This is to ensure prompt delivery of responses to different nodes and yet combine answer messages directed to the same node into the same frame for greater network efficiency.

The Update Task flushes all queued messages to the network after it has processed all active requests each 15 Hz cycle.

D0 Settings

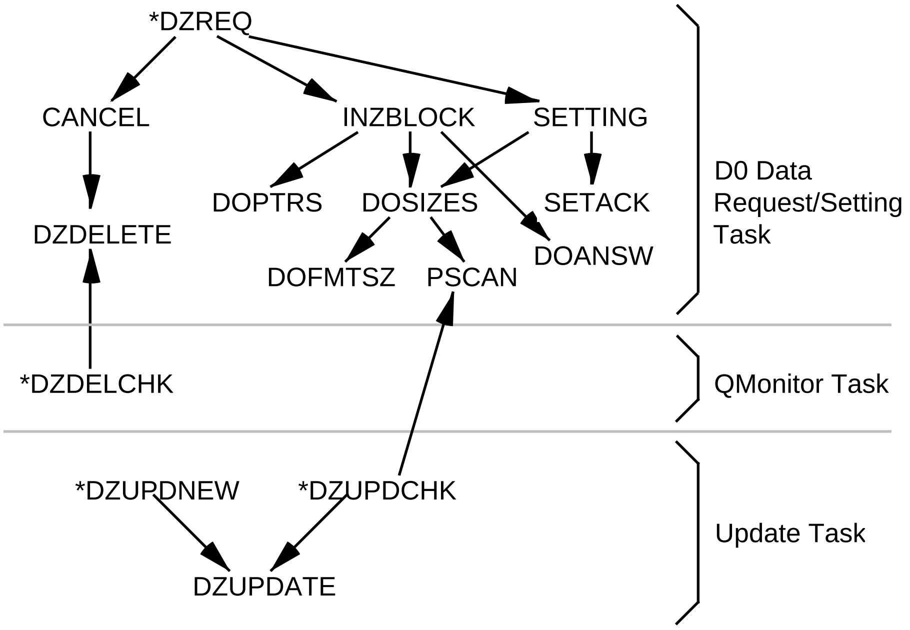
Processing setting messages is greatly simplified because it is all done immediately and because the format of the setting message is nearly identical with that of the request message. The message type word is different to indicate that is is a setting, the period specification is absent, and the setting data offset is specified in the first three words of the message header.

The many set-type routines have been enhanced so that they now return error codes whenever they encounter errors. (Previously, the setting was simply ignored.) This error response word is used in the setting acknowledge message specified in the protocol. A zero value indicates no detected error in performing the setting.

Since setting processing includes an overview scan of the validity of the message, performed by `DOSIZES`, a status-only reply may be given to a setting in place of the setting acknowledge message. For the status-only cases as well as the setting acknowledge cases, refer to the error codes given in the “Error reporting” section of this document.

D0 Request Module Road Map

The organization of the routines in the DZREQ module is as follows, where an asterisk denotes a declared entry point:



The upper collection of routines comprise the DZero Request Task, which waits for a message directed to the destination taskname RPYR and processes it. For a *request* message, the CANCEL routine searches the active list chain for a match against the message id (“list#”), the requesting node and source task id. If it finds a match, it calls DZDELETE to cancel that active request. The INZBLOCK is the bulk of the code which prepares the request block, internal pointers block and answers block for later processing by the Update Task. It uses several other routines to help break that job down into more manageable pieces.

For a *setting* message, the DOSIZES routine is invoked to check for a number of obvious errors. If an error is detected, a status-only reply is given. If not, a doubly-nested loop—outer loop over listypes, inner loop over idents—calls the system routine SETLOCAL to process each setting listype/ident pair. An error return aborts the processing of any remaining settings in the message, and SETACK is invoked to deliver the setting acknowledgment message.

The middle section is the DZDELCHK routine which is called by the QMonitor Task when it has detected the completion of transmission of an Acnet-type message (block type#9) with bit#6 of the NetQFlg word set in the block, indicating that the block is to be retained for re-use. (If the bit were not set, QMonitor would simply free the memory for that block.) It checks for the case of a one-shot DZero data request that should be cancelled. So QMonitor has to the `NetQFlg` was set indicating that the block was queued for transmission to the network.

The last section includes two entry points that are called by the Update Task to process type#13 requests during its traversal of the active request chain. `DZUPDNEW` updates the request only if it has never been updated before, whereas `DZUPDCHK` examines the period specification and updates the request only if it is time for an update. `DZUPDATE` shepherds the actual updating of the request and checks the blocking parameters before queuing a response to the network.

Error reporting for requests

A number of potential errors are detected when processing a D0 data request message. For most of these, a response is returned to the requester consisting of a status-only reply, which includes only the Acnet header; neither the format block nor the answer message is attached. Current error codes are as follows:

-64	period spec not implemented yet
-65	invalid message size
-66	invalid request header size
-67	invalid DZero message type
-68	invalid #listypes
-69	dynamic memory unavailable
-70	invalid listype#
-71	invalid identype (error in listype table)
-72	invalid ident length for listype#
-73	invalid #idents for single listype
-74	invalid #bytes requested per ident
-75	invalid offset to ident array in LCB
-76	format block / #bytes conflict
-77	requested #bytes exceeded format spec template
-78	invalid total #idents this request
-79	size of answers format block too large
-80	size of answers too large
-81	#sets of answers too large (blocking spec)
-82	invalid format spec (error in listype table)
-83	request message data offset not implemented
-84	LCB “other” parameters not implemented
-85	spare
-86	setting message data offset out of range
-87	setting message included period spec

In addition to the response to the requester, these errors are recorded in the Local Station in local variables of the DZero Request Task. They can be inspected for

Another error that can be returned by the Network Layer itself is the following:

-21 destination task not connected to network (RPYR not connected)

This means that the 4-byte destination task name in the Acnet header was not recognized by the node that received it. For systems which have Network Layer support but have not yet been updated with the D0 data request software, this will certainly result.

Setting acknowledgment error codes

The following list of errors can occur in response to a data setting message:

0	No error. Setting successful.
1	System table not defined for this listype.
2	Entry# (chan#, bit#, etc) out of range.
3	Odd #bytes of data
4	Bus error
5	#bytes too small
6	#bytes too large
7	Invalid #bytes
8	Set-type out of range (error in listype table)
9	Settings not allowed for this listype
10	Analog control type# out of range (error in analog descriptor)
11	Invalid binary byte address in BADDR table
12	Invalid mpx channel# (Linac D/A hardware)
13	F3 scale factor out of range (motor #steps processing)
14	No CPROQ table or co-proc# out of range
15	Hardware D/A board address odd
16	Bit# index out of range (associated bit control via channel)
17	Bit# out of range for this system's database
18	Digital Control Delay table full (for software-formed pulses)
19	Digital control type# out of range 1–15
20	Co-processor command queue unavailable
21	Co-processor invalid queue header
22	Queue full or unavailable
23	Dynamic memory allocation failed
24	Error status from 1553 controller
25	Invalid 1553 command for one word output
26	Invalid 1553 Command Block address (must be multiple of 16)
27	Invalid 1553 order code in first word of Command Block
28	1553 interrupts not working
29	Cannot initialize 1553 command queue
30	No Q1553 table of pointers to 1553 controller queues
31	Invalid Motor table

35	Invalid data value.
36	Invalid #bytes of text in Comment alarm control
37	No DSTRM table of Data Stream queue pointers
38	Data Stream queue type# out of range
39	Data Stream queue not initialized
40	No MMAPS table of memory-mapped board templates
41	Invalid MMAPS table header
42	Invalid MMAPS table entry size
43	Invalid board# for MMAPS table
44	Invalid directory entry in MMAPS table
45	End of MMAPS table reached during template processing
46	Invalid MMAPS command type code
47	Invalid MMAPS loop params
48	Invalid MMAPS nested loop
49	spare
50	Invalid listype#
51	Invalid ident type# (error in listype table)
52	Invalid ident length for this listype
53	Little console settings switch disabled
54	Little console external settings switch disabled
55	Data Server setting not implemented

Limitations of present implementation

Features *not* supported in the initial version of DZero request handling are the following:

Period specifications *besides* one-shots and simple periodic and blocking
Data offset specified at listype level
"Other" parameters specified at listype level
Error status reporting for each listype-ident pair

It is not intended to support data requests of the "Data Server" type for the D0 protocol. Idents in a request are *ignored* if they do not include the node# of the local station receiving the request in the first word of the ident. This means that one could send the same request to a group of nodes using the functional group multicast form of network addressing, and each node receiving the request would select out its own idents for answer response. (Obviously the requesting node would need to scan the original request in order to be in a position to match the answers with the questions.) Currently, however, the Acnet header-based protocols do not permit sending request messages to a group of nodes.